

From PeleC to PeleACC, to PeleC++

P3HPC Forum 2020, September 2
Jon Rood, Marc Henry de Frahan, Ray
Grout

The Pele Project

Solves reacting Navier-Stokes on structured grid using AMR and embedded boundaries based on AMReX library

PeleC

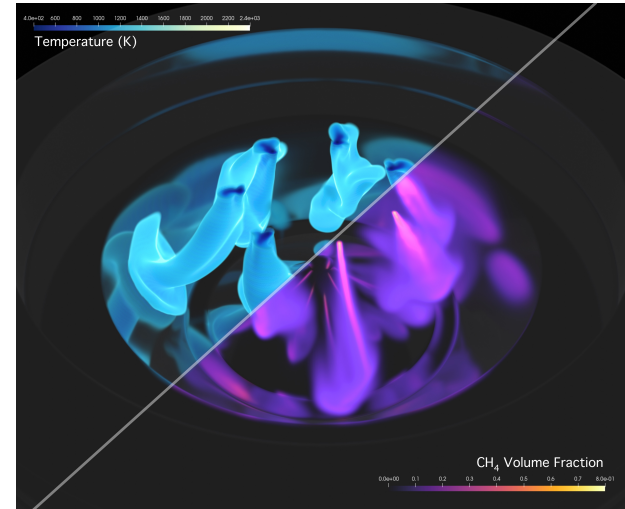
- Compressible combustion simulations
- Explicit time stepping

PeleLM

- Low-mach combustion simulations
- Implicit, requiring linear solver

PelePhysics

- Shared code for chemistry/reactions



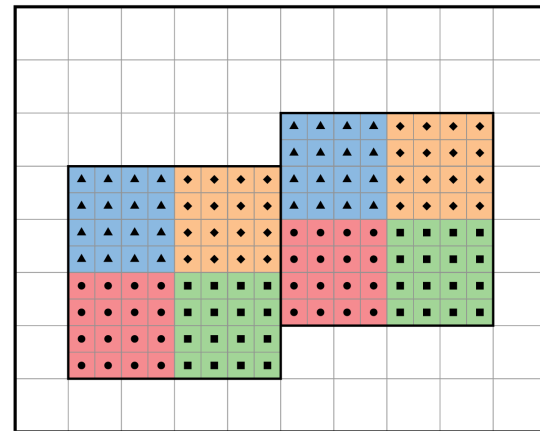
PeleC Overview

- 50k LOC
- 11373 lines of C++
- 38905 lines of Fortran (including duplicate dimension-specific code)
- High level C++ orchestration with Fortran kernels
- Source code generator used for chemistry to unroll code
- C++ -> Fortran -> C
 - Mixed languages pose many issues

Original PeleC Programming Model

- MPI + OpenMP
- Ranks operate in bulk-synchronous data-parallel fashion
- Threads operate on independent tiles
- Originally focused on KNL and vectorization (lowered loops)

```
#pragma omp parallel
for (MFIter mfi(F,true); mfi.isValid(); ++mfi) {
    const Box& box = mfi.tilebox();
    Array4<Real const> const& u = U.const_array(mfi);
    Array4<Real          > const& f = F.array(mfi);
    f2(box, u, f); // Call Fortran kernel
}
```



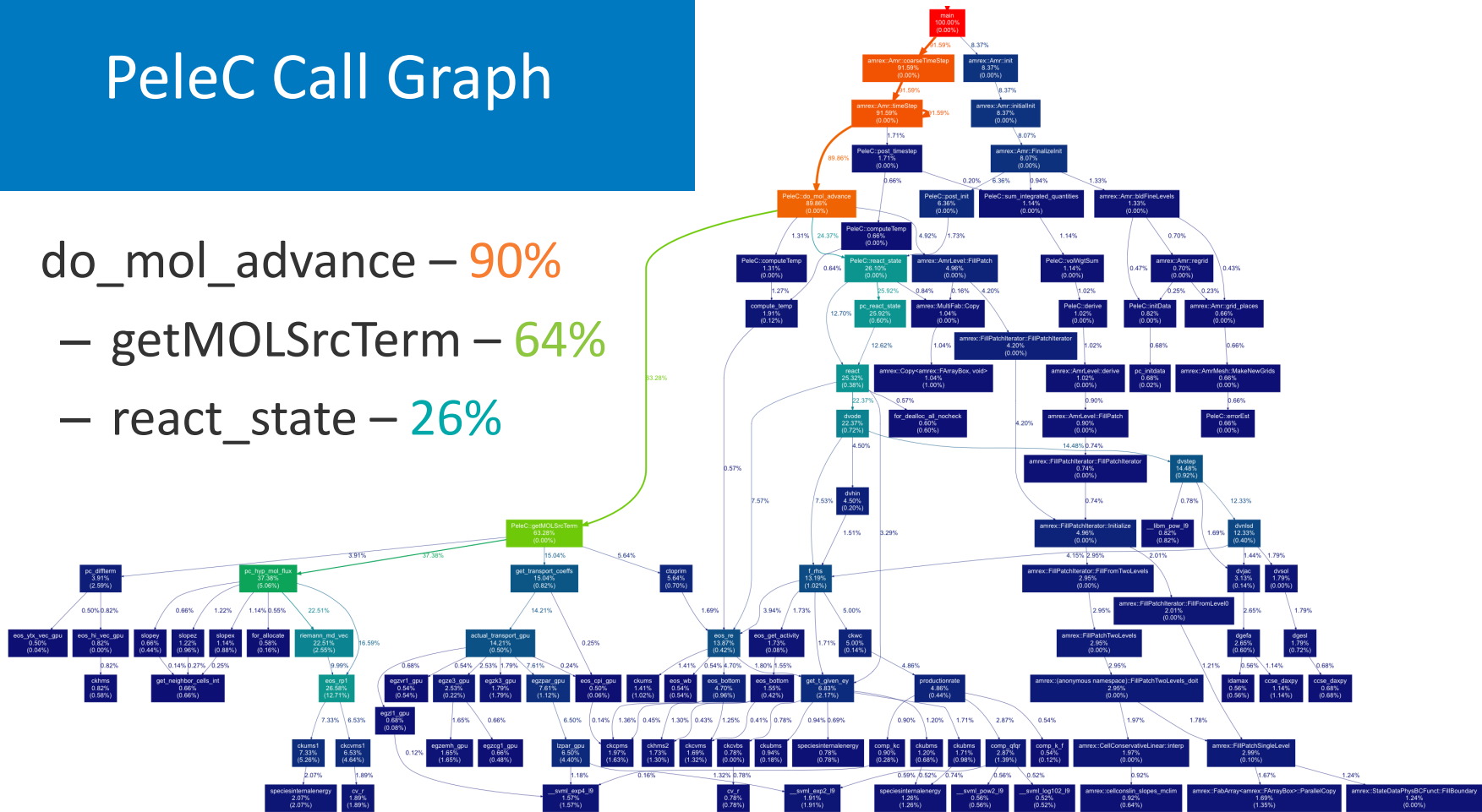
AMReX FAB data structures¹.

PeleC on GPUs

- Xeon Phi discontinued; GPUs become focus for birth of Exascale
- Quickest way to utilizing GPUs
 - Offload kernels to device
- OpenACC most mature Fortran GPU programming model at the time
- Tied to PGI compiler
- Introduced in 2011
 - Used in production since ~2014
- OpenMP 4 introduced for accelerators in 2013
 - Jeff Larkin (NVIDIA) - GTC **March 2018** – OpenMP on GPUs, **First Experiences** and Best Practices
- OpenACC pragmas have a straightforward mapping to OpenMP pragmas
- Minimize the need to modify current PeleC code
- Don't need to remove current OpenMP pragmas

PeleC Call Graph

- do_mol_advance – 90%
 - getMOLSrcTerm – 64%
 - react_state – 26%



OpenACC Effort

- 90% of runtime under one routine
- Around 5 kernel routines under `getMOLSrcTerm` to parallelize on GPU
 - Around 50 routines to label as `seq`
 - Wrote Fortran version of Fuego code generator for these routines
- `react_state` is implicit ODE solver with thousands of if conditions
 - Implement a simpler explicit solver instead
 - Explicit solver written in C and CUDA
 - Explicit solver 6x slower on CPU
 - Completely dominates runtime (`react_state` now around 90%)

PeleC OpenACC Programming Model

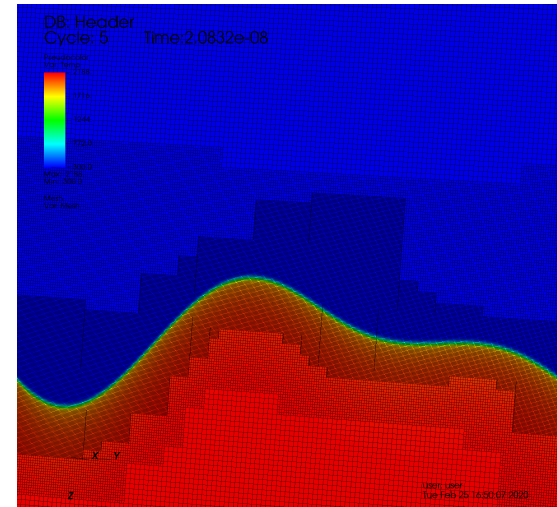
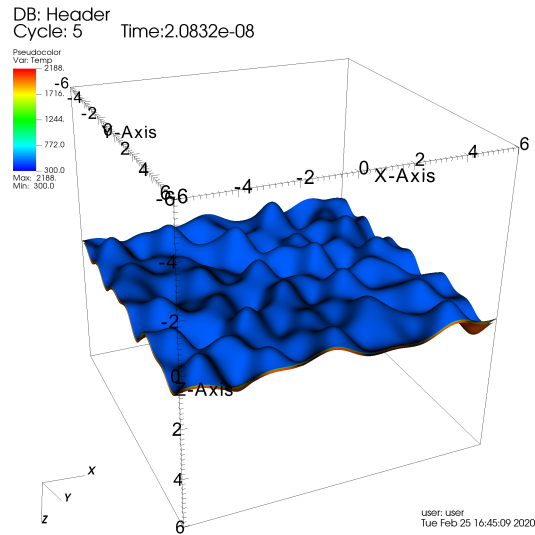
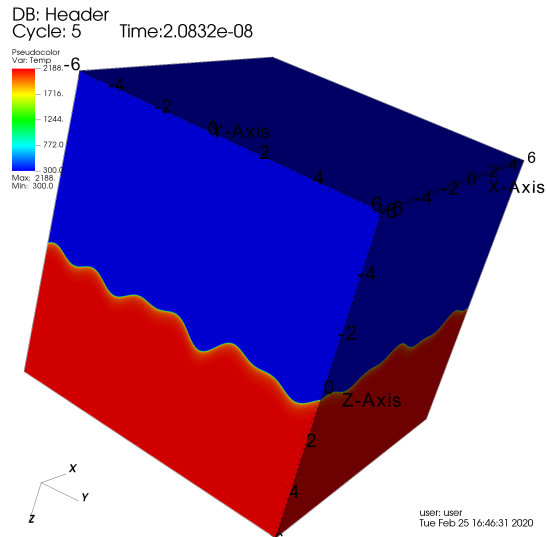
- Memory management originally done explicitly
- Later used AMReX's GPU memory management
 - Use default (present)
- Just need to make sure every routine under kernel is decorated as seq device routine
- Run with MPS, 7 ranks per Summit GPU to obtain asynchronous kernels

```
for (MFIter mfi(mf,TilingIfNotGPU()); mfi.isValid(); ++mfi)
{
    const Box& bx = mfi.tilebox();
    FArrayBox& fab = mf[mfi];
    plusone_acc(BL_TO_FORTTRAN_BOX(tbx),
               BL_TO_FORTTRAN_ANYD(fab));
}

subroutine plusone_acc()
!$acc parallel loop gang vector collapse(3) default(present)
do k = lo3, hi3
    do j = lo2, hi2
        do i = lo1, hi1
            data(i,j,k) = data(i,j,k) + 1.0_amrex_real
            call deep_nest_of_functions()
        end do
    end do
end do
!$acc end parallel loop
end subroutine plusone_acc
```

Figure 2: OpenACC approach to launching a kernel on the GPU.

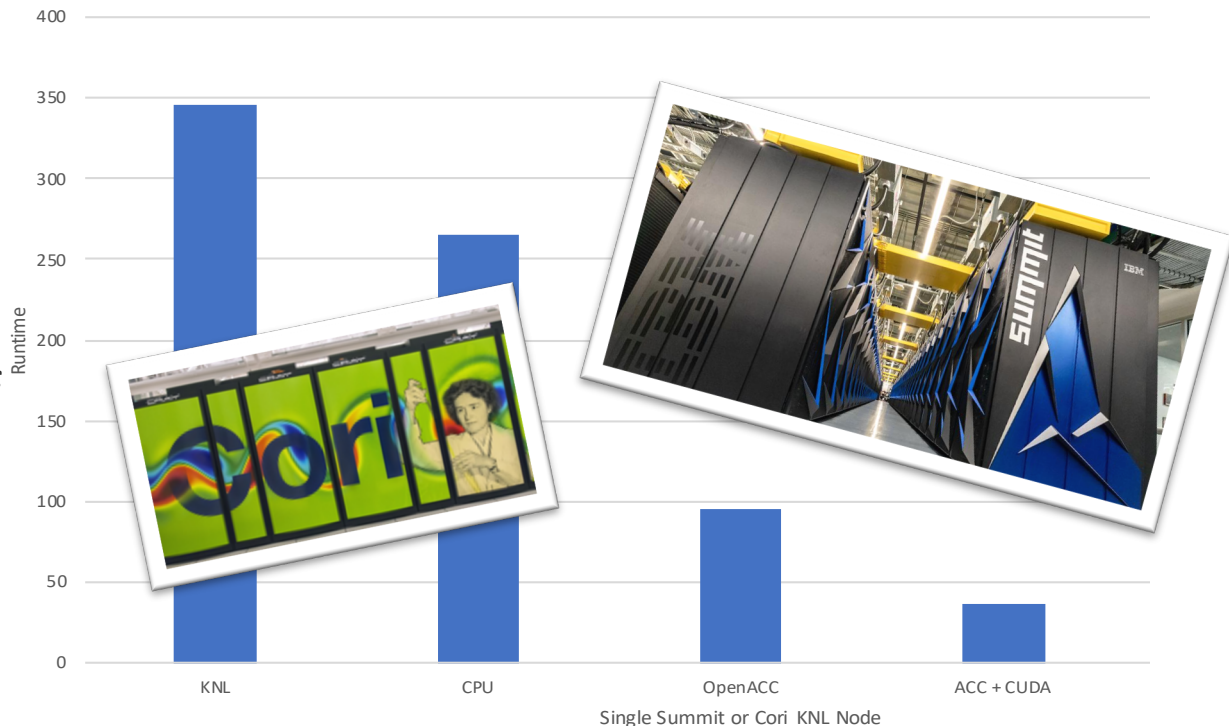
Test Case – Pre-mixed Flame



OpenACC Results

- Initial OpenACC port over 3x faster than Cori KNL
- 8x faster with CUDA `react_state()`
- 2 people, 5 weeks of development time
- 1 major bug found and reported to PGI

PeleC GPU Port - Summit vs Cori Node - PMF 3D Case



C++ Effort

- AMReX GPU strategy was emerging alongside our OpenACC effort
 - Much like Kokkos using C++ lambdas, but need not be as general
- Steven Reeves, graduate student at LBL prototyped PeleC on the GPU by porting every necessary routine to C++
 - Performance much better than OpenACC prototype
- However, once AMReX's memory management was used in OpenACC, performance over OpenACC seemed to be a toss-up (mostly due to sharing of `react_state` routine)
- Performance in general was 16-18x faster than KNL

OpenACC vs C++ Prototype

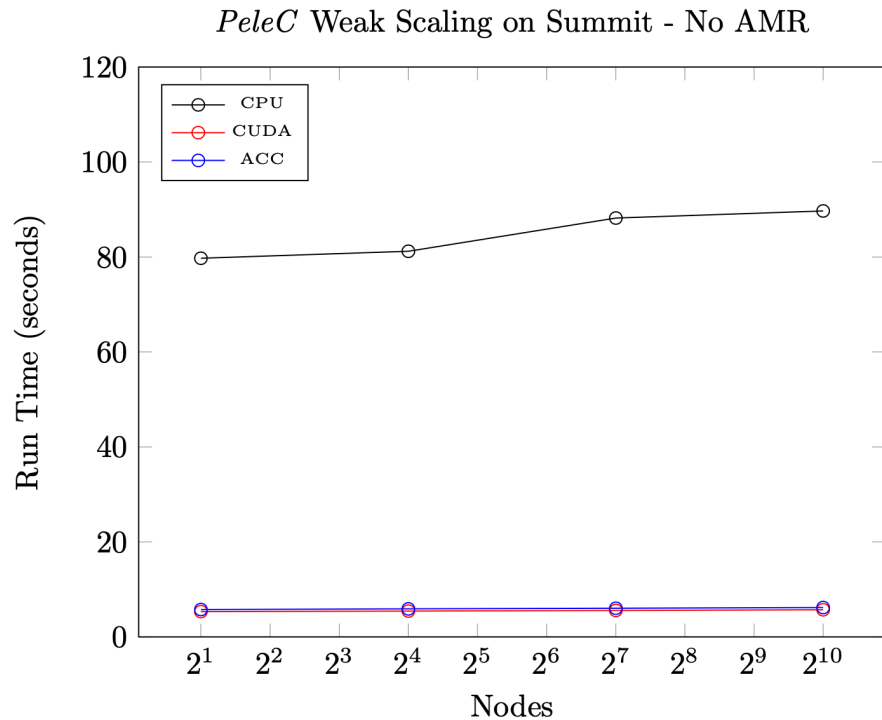
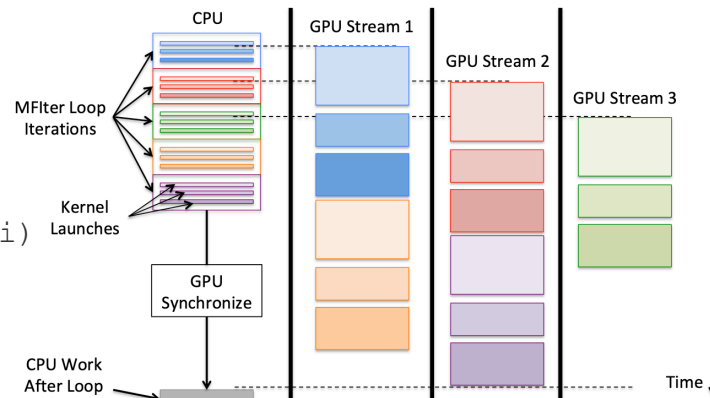
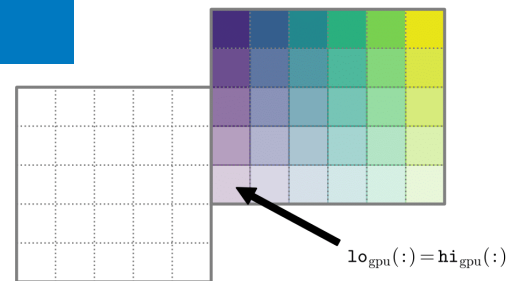


Figure 3: Weak scaling of PMF problem with 2^{23} cells per node and no AMR.

C++ Effort

- MPI+CUDA for GPUs
- Essentially one thread per cell
- Focus on maximum parallelism in kernel (hoisted perfectly nested loops)
- 1 rank per GPU with CUDA streams for asynchronous behavior

```
#pragma omp parallel if (amrex::Gpu::notInLaunchRegion())
for (MFIter mfi(mf,TilingIfNotGPU()); mfi.isValid(); ++mfi)
{
    const Box& bx = mfi.tilebox();
    Array4<Real> const& fab = mf.array(mfi);
    amrex::ParallelFor(bx, ncomp,
    [=] AMREX_GPU_DEVICE (int i, int j, int k, int n)
    {
        fab(i,j,k,n) += 1.;
    });
}
```

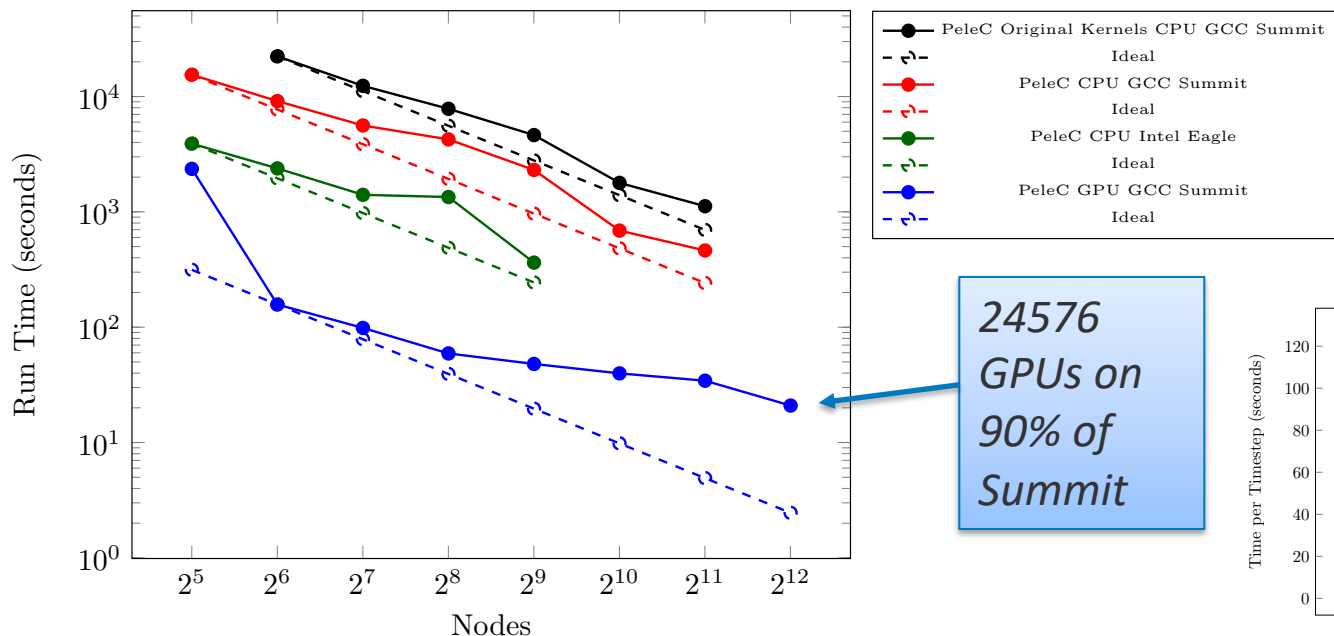


AMReX GPU strategy².

C++ Results

- 2x faster on CPU
- 18x faster than fastest CPU case using Intel compiler
- 56x faster than GCC CPU on Summit
- 124x faster than original Fortran on Summit CPUs

PeleC Strong Scaling on Summit and Eagle



24576
GPUs on
90% of
Summit

PeleC Weak Scaling on Summit GPUs

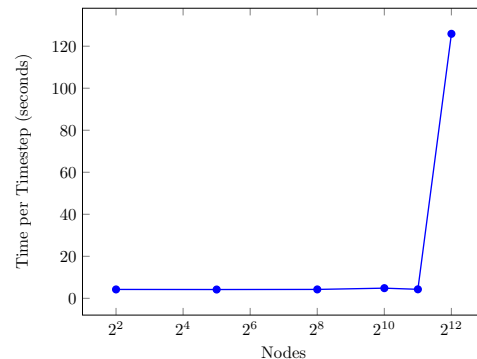


Figure 1: Weak scaling of PMF case with drm19 chemistry with no AMR. 2²² cells per node.

Figure 1: Strong scaling of PMF case with drm19 chemistry on Summit and Eagle machines. 360M cells with 2 levels of AMR.

Conclusions

- OpenACC allowed us to prototype PeleC on GPU very quickly
- Performance can be similar to CUDA
- Code quickly became displeasing
- Mixed languages cause problems for readability, debugging, profiling, and compiler optimizations
- Non-ubiquitous programming models lack support, robustness, and flexibility
- Fortran was holding us back
- PeleC now 19363 lines of C++
- Fortran appears to be not beneficial to PeleC in any way
- Even 2x faster on the CPU
- Easier to debug and profile
- Kernels easier to write and to read
- Much less duplicate code necessary for dimensions
- Ability to use many compilers
- Good performance portability
- 1 graduate student 6 months + 2 staff 12 weeks to completely move to C++

References

1. https://amrex-codes.github.io/amrex/docs_html/Basics.html#mfiter-and-tiling
2. https://amrex-codes.github.io/amrex/docs_html/GPU.html#overview-of-amrex-gpu-strategy

Q&A

www.nrel.gov

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

This work was authored by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by the Exascale Computing Project. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

